What is claimed is:

1. A computer system comprising:

a memory;

a central processing unit (CPU) configured to read and write data to the memory; and

a graphics processing unit (GPU) configured to at least read data from the memory wherein the GPU is configured to execute applications in a multithreaded manner and provide each executing application with its own virtual address space into the memory.

2. The computer system of claim 1 wherein the GPU is configured to select which application to execute next based on a run list.

3. The computer system of claim 2 wherein the CPU is configured to maintain the run list.

4. The computer system of claim 1 wherein the GPU uses a handle table to manage paging of the virtual address space.

5. The computer system of claim 1 wherein the GPU uses a handle table with memory descriptor list handles to manage paging of the virtual address space.

6. The computer system of claim 1 wherein the GPU uses a flat page table to manage paging of the virtual address space.

7. The computer system of claim 1 wherein the GPU uses a multilevel page table to manage paging of the virtual address space.

8. The computer system of claim 1, further comprising a GPU scheduler wherein the GPU scheduler is configured to track two current GPU contexts in a run wave.

9. The computer system of claim 1 wherein the GPU is configured to perform demand faulting for data needed by the GPU that is not loaded.

10. The computer system of claim 9 wherein the level of faulting is at a surface granularity.

11. The computer system of claim 10 wherein a page fault is generated in response to a context switch occurring to a context that references an invalid buffer.

12. The computer system of claim 10 wherein a page fault is generated in response to an instruction to the GPU to draw a primitive for which some or all of the required resources are not loaded.

13. The computer system of claim 1 wherein the GPU supports both a limited DMA buffer and a privileged DMA buffer.

14. The computer system of claim 1 wherein the GPU supports a programmable PCI aperture.

15. A method for scheduling tasks for processing by a coprocessor, comprising:

gathering tasks for processing by a coprocessor into a memory group wherein the memory group relates to a first application;

delivering the tasks to a scheduler wherein scheduler functions include determining an order for processing the tasks wherein the order may include tasks that relate to one or more other applications;

determining an order for processing the tasks wherein the order accounts for any relative priority among the first application and one or more other applications and a corresponding amount of processing time that the first application and one or more other applications are entitled to;

preparing tasks for processing by ensuring that any needed memory resources are available in a coprocessor-accessible memory location wherein the preparing tasks occurs in the order determined by the scheduler; and

submitting tasks prepared according to the preparing to the coprocessor for processing.

16. A method according to claim 15 wherein the coprocessor includes a graphics processing unit (GPU).

17. A method according to claim 15, further comprising calling an Application Program Interface (API) when the first application has one or more tasks that require processing by the coprocessor.

18. A method according to claim 17, further comprising calling a user mode driver wherein the functions of the user mode driver include placing rendering commands associated with the one or more tasks in the memory group.

19. A method according to claim 18, further comprising returning the rendering commands to the API, and submitting them to a coprocessor kernel.

20. A method according to claim 15, further comprising generating a Dynamic Memory Access (DMA) buffer by a kernel mode driver wherein one or more tasks that require processing by the coprocessor are used to generate the DMA buffer, and the DMA buffer represents the one or more tasks used to generate the DMA buffer.

21. A method according to claim 20, further comprising generating a list of memory resources by the kernel mode driver wherein the memory resources represented by the list are needed by the coprocessor to process one or more tasks represented by the DMA buffer.

22. A method according to claim 21, further comprising building a paging buffer for bringing the memory resources on the list of memory resources to correct memory addresses within the coprocessor-accessible memory location.

23. A method according to claim 15 wherein said preparing is accomplished by a preparation thread which calls a memory manager process capable of determining a location in the coprocessor-accessible memory location to page any needed memory resources.

24. A method according to claim 23, further comprising splitting a DMA buffer when the memory manager process determines that there is not enough room in the coprocessor-accessible memory location to page all needed memory resources.

25. A computer readable medium comprising computer executable instructions for carrying out the method of claim 15.

26. A modulated data signal carrying computer executable instructions for use in performing the method of claim 15.

27. A computing device comprising means for performing the method of claim 15.

28. A method for scheduling tasks for processing by a coprocessor, comprising:

gathering tasks for processing by a coprocessor into a memory group wherein the memory group relates to a first application;

delivering the tasks to a scheduler wherein the functions of the scheduler include determining an order for processing the tasks wherein the order may include tasks that relate to one or more other applications;

determining an order for processing the tasks wherein the order accounts for any relative priority among the first application and one or more other applications and a corresponding amount of processing time that the first application and one or more other applications are entitled to;

preparing tasks for processing by ensuring that any needed memory resources are available in a memory location accessible by the coprocessor wherein the preparing tasks occurs in the order determined by the scheduler;

submitting tasks to the coprocessor for processing;

managing the coprocessor-readable memory to apportion the coprocessor-readable memory among the various tasks; and

providing a virtual address space for the tasks.

29. A method according to claim 28 wherein the coprocessor is a graphics processing unit (GPU).

30. A method according to claim 28, further comprising storing a task in a DMA buffer wherein the storing is accomplished by a user mode driver.

31. A method according to claim 30, further comprising validating a memory resource referenced in a resource list that is associated with the DMA buffer wherein validating entails finding a range of coprocessor-readable memory that is free and asking the kernel mode driver to map a page table or a memory resource handle to that range.

32. A method according to claim 28 wherein the virtual address space is virtualized through the use of a flat page table that divides coprocessor-readable memory into pages of a predefined memory amount wherein further a page table is provided in the virtual address space that contains identifiers for specifying coprocessor-readable memory addresses.

33. A method according to claim 28 wherein the virtual address space is virtualized through the use of a multi-level page table that divides coprocessor-readable memory into pages of a predefined memory amount wherein further a multiple page tables are provided in the virtual address space that contain identifiers for specifying coprocessor-readable memory addresses.

34. A method according to claim 28 wherein a portion of coprocessor readable memory is used to indicate whether all required memory resources associated with a task that requires processing are available in coprocessor-readable memory.

35. A computer readable medium comprising computer executable instructions for carrying out the method of claim 28.

36. A modulated data signal carrying computer executable instructions for use in performing the method of claim 28.

37. A computing device comprising means for performing the method of claim 28.

38. A method according to claim 28, further comprising:

assigning a base address for a display surface wherein the display surface is allocated contiguously in coprocessor-readable memory; and

delivering a task to the scheduler wherein processing the task will reassign the base address for a display surface.

39. A method according to claim 38 wherein processing the task will reassign the base address for a display surface immediately.

40. A method according to claim 38 wherein processing the task will reassign the base address for a display surface upon the occurrence of a subsequent display synchronization period.

41. An apparatus for supporting scheduling of tasks for processing by a coprocessor, comprising:

a central processing unit (CPU);

a coprocessor;

one or more applications that generate tasks for processing by the coprocessor wherein the tasks are first stored in an application-specific memory location;

a scheduler process for determining an order in which the tasks are processed; wherein the order accounts for any relative priority among a first application and one or more other applications and a corresponding amount of processing time that the first application and one or more other applications are entitled to.

42. An apparatus according to claim 41 wherein the coprocessor is a GPU.

43. An apparatus according to claim 41 wherein the coprocessor supports interruption during the processing of a task by automatically saving task information to a coprocessor-accessible memory location.

44. An apparatus according to claim 43, further comprising at least one of a private address space for one or more tasks, a private ring buffer where tasks are accumulated, and a private piece of coprocessor-accessible memory where a hardware state is saved when a task is not being processed.

45. An apparatus according to claim 41 wherein the coprocessor is capable of storing information regarding the history of coprocessor switches from task to task in a specified system memory location readable by the scheduler process.

46. An apparatus according to claim 45 wherein the coprocessor specifies a base address for the system memory location prior to storing information regarding the history of coprocessor switches from task to task in the system memory location.

47. An apparatus according to claim 45 wherein the coprocessor specifies a size for the system memory location prior to storing information regarding the history of coprocessor switches from task to task in the system memory location.

48. An apparatus according to claim 45 wherein the coprocessor specifies a write pointer for indicating where in the system memory location the coprocessor should write to next.

49. An apparatus according to claim 41 wherein the coprocessor supports fence instructions that cause the coprocessor to write a piece of data associated with a fence instruction at an address specified in the fence instruction.

50. An apparatus according to claim 41 wherein the coprocessor supports trap instructions that are capable of generating a CPU interrupt when processed by the coprocessor.

51. An apparatus according to claim 41 wherein the coprocessor supports enable/disable context switching instructions such that when context switching is disabled, the coprocessor will not switch away from a current coprocessor task.

52. A method for providing applications with memory to support the processing of tasks for processing by a coprocessor, comprising:

providing a virtual address space to at least one application;

storing information relating to one or more tasks for processing by a coprocessor in the virtual address space wherein the one or more tasks are generated at least in part by the at least one application;

identifying a location in physical memory that corresponds to at least one virtual address in the virtual address space;

accessing the location in physical memory that corresponds to at least one virtual address in the virtual address space when the one or more tasks are submitted to the coprocessor for processing.

53. A method according to claim 52 wherein the coprocessor includes a graphics processing unit (GPU).

54. A method according to claim 52 wherein the identifying a location in physical memory is accomplished by a memory manager that can move memory resources to another location in physical memory.

55. A method according to claim 52 wherein the information relating to one or more tasks is assigned to separate portions of the virtual address space, each portion comprising context information that includes the location of memory resources required to process the task.

56. A method according to claim 52, further comprising validating a memory resource wherein the validating comprises finding a range of physical memory that is free and requesting a driver to map a memory resource handle to that range.

57. A method according to claim 52 wherein the virtual address space is virtualized through the use of a flat page table that divides coprocessor-readable memory into pages of a predefined memory amount wherein further a page table is provided in the virtual address space that contains identifiers for specifying coprocessor-readable memory addresses.

58. A method according to claim 52 wherein the virtual address space is virtualized through the use of a multi-level page table that divides coprocessor-readable memory into pages of a predefined memory amount wherein further a multiple page tables are provided in the virtual address space that contain identifiers for specifying coprocessor-readable memory addresses.

59. A method according to claim 52 wherein a portion of the physical memory is used to indicate whether all required memory resources associated with a task that requires processing are available for processing in physical memory.

60. A method according to claim 52 wherein the physical memory comprises two portions, a large portion and a small portion, and the small portion includes references to memory locations in the large portion.

61. A method according to claim 60 wherein the small portion references four kilobyte (6) blocks of memory in the large portion.

62. A method according to claim 60 wherein a supplemental memory manager maps the references to memory locations in the large portion.

63. A computer readable medium comprising computer executable instructions for carrying out the method of claim 52.

64. A modulated data signal carrying computer executable instructions for use in performing the method of claim 52.

65. A computing device comprising means for performing the method of claim 52.

66. A coprocessor for use in connection with a coprocessing scheduler, comprising:

a coprocessor for processing tasks that are submitted to the coprocessor by a scheduler process wherein the scheduler process submits tasks to the coprocessor according to a priority of applications that request processing of the tasks, and wherein the priority determines the amount of coprocessor time one or more applications are entitled to.

67. A coprocessor according to claim 66 wherein the tasks are first stored in an application-specific memory location.

68. A coprocessor according to claim 66 wherein the coprocessor stores information related to a task in a per-context address space, and wherein further the information related to a task allows the coprocessor to process the task or a portion of the task after processing one or more intervening tasks.

69. A coprocessor according to claim 66 wherein the coprocessor processes tasks from a run list by switching immediately to a subsequent task on the run list when a switching event occurs.

70. A coprocessor according to claim 69 wherein a switching event comprises at least one of a completion of processing a previously submitted task, a page fault in processing a task, a general

protection fault in processing a task, and a request by a central processing unit (CPU) to switch to a new run list.

71. A coprocessor according to claim 66 wherein the coprocessor comprises a GPU.

72. A coprocessor according to claim 66 wherein the coprocessor accesses memory resources in a coprocessor-readable memory by a memory manager.

73. A coprocessor according to claim 72 wherein the memory resources comprise references to virtual memory addresses.

74. A computing system that enables the efficient scheduling of coprocessor tasks by allowing a user mode driver to build DMA buffers without compromising system security, comprising:

a coprocessor;

memory that is designated as privileged memory;

a user mode driver that builds a limited DMA buffer wherein the coprocessor cannot access the privileged memory when processing the limited DMA buffer; and

a kernel mode that builds a privileged DMA buffer wherein the coprocessor can access the privileged memory when processing the privileged DMA buffer.

75. An apparatus according to claim 74 wherein the coprocessor is a GPU.

76. An apparatus according to claim 74, further comprising one of a mechanism to specify on a per-handle basis whether memory is designated as privileged memory and a mechanism to specify on a per-page basis whether memory is designated as privileged memory.

77. An apparatus according to claim 74 wherein the coprocessor generates a page fault if the limited DMA buffer instructs it to access memory that is designated as privileged memory.

78. An apparatus according to claim 74 wherein features of the limited DMA buffer comprise one of containing only references to virtual addresses, not containing instructions that would affect a current display, not containing instructions that would affect an adapter, having limited power management, having limited config space, and not containing instructions that prevent context switching.

79. An apparatus according to claim 74 wherein the memory that is designated as privileged memory is designated as privileged on a per-coprocessor context basis.